The GriPhyN Virtual Data System Properties

Property Documentation

automatically generated at

2005-01-25 13:33

Contents

1 Introduction

This file will report the majority of the settable properties, and their respective default values. Please refer to the user guide for a more in-depth discussion of the configuration options. Please note that the values rely on proper capitalization, unless explicitely noted otherwise.

Some properties rely with their default on the value of other properties. As a notation, the curly braces refer to the value of the named property. For instance, ${vds.home} means that the value depends on the value of the vds.home property plus any noted additions. You can use this notation to refer to other properties, though the extend of the subsitutions are limited. Usually, you want to refer to a set of the standard system properties. Nesting is not allowed. Substitutions will only be done once.

There is a priority handling to properties. By default, the average user does not need to worry about the priorities. However, it is good to know the details of when which property applies, and how one property is able to overwrite another.

1. Property definitions in the system property file, usually found in ${vds.home.sysconfdir}/properties, have the lowest priority. These properties are expected to be set up by the submit host's administrator.

2. The properties defined in the user property file ${user.home}/.chimerarc have higher priority. These can overwrite settings found in the system's properties. A set of sensible property values to set on a production system is shown below.

3. Commandline properties have the highest priority. Each commandline property is introduced by a -D argument. Note that these arguments are parsed by the shell wrapper, and thus the -D arguments must be the first arguments to any command. Commandline properties are useful for debugging purposes.

The following example provides a sensible set of properties to be set by the user property file. These properties use mostly non-default settings. It is an example only, and will not work for you:

```
vds.db.vdc.schema        ChunkSchema
vds.db.ptc.schema        InvocationSchema
vds.db.*.driver          Postgres
vds.db.*.driver.url      jdbc:postgresql:${user.name}
vds.db.*.driver.user     ${user.name}
vds.db.*.driver.password XXXXXXXX
vds.rls.url              rls://sheveled.mcs.anl.gov
vds.replica.mode         rls
vds.exitcode.mode        all
vds.transfer.mode        single
vds.pool.mode            single
vds.pool.file            ${vds.home}/contrib/Euryale/Grid3/pool.config
```

If you are in doubt which properties are actually visible, a sample application called `testprops` dumps all properties after reading them.

2 Basics

2.1 vds.home

Systems	all
Type	directory location string
Default	""$VDS_HOME""

The property vds.home cannot be set in the property file. Any of the shell wrapper script for the applications will set this property from the value of the environment variable $VDS_HOME.

3 Directory Location Properties

3.1 vds.properties

Systems	all
Type	file location string
Default	${vds.home.sysconfdir}/properties

The system-wide properties file will be looked for in its default place. It will usually reside in $VDS_HOME/etc as file named properties.

3.2 vds.user.properties

Systems	all
Type	file location string
Default	${user.home}/.chimerarc

Each user can overwrite the system-wide properties with his or her own definitions. The user properties rely on the system's notion of the user home directory, as reflected in the JRE system properties. In the user's home directory, a file .chimerarc will be taken to contain user property definitions. Note that ${user.home} is a system property.

3.3 vds.home.datadir

Systems	all
Type	directory location string
Default	${vds.home}/share

The datadir directory contains broadly visiable and possilby exported configuration files that rarely change. This directory is currently unused.

3.4 vds.home.sysconfdir

Systems	all
Type	directory location string
Default	${vds.home}/etc

The system configuration directory contains configuration files that are specific to the machine or installation, and that rarely change. This is the directory where the XML schema definition copies are stored, and where the base pool configuration file is stored.

[NOTE to Gaurang: the dyanmic poolcfg xml file should be in localstate]

3.5 vds.home.sharedstatedir

Systems	all
Type	directory location string
Default	${vds.home}/com

Frequently changing files that are broadly visible are stored in the shared state directory. This is currently unused.

3.6 vds.home.localstatedir

Systems	all
Type	directory location string
Default	${vds.home}/var

Frequently changing files that are specific to a machine and/or installation are stored in the local state directory. This directory is being used for the textual transformation catalog, file-based VDCs, and the file-based replica catalog of the shell planner.

4 Simple File Location Properties

4.1 vds.schema.vdl

Systems	Chimera, VDC
Type	XML schema file location string
Default	${vds.home.sysconfdir}/vds-1.21.xsd

This file is a copy of the XML schema that describes VDLx files. VDLx files and fragments are used in various places throughout the abstract planning process. Providing a copy of the schema enables the parser to use the local copy instead of reaching out to the internet, and obtaining the latest version from the GriPhyN website dynamically.

4.2 vds.schema.dax

Systems	all
Type	XML schema file location string
Value[0]	${vds.home.sysconfdir}/dax-1.5.xsd
Value[1]	${vds.home.sysconfdir}/dax-1.6.xsd
Value[2]	${vds.home.sysconfdir}/dax-1.7.xsd
Default	${vds.home.sysconfdir}/dax-1.7.xsd

This file is a copy of the XML schema that describes abstract DAG files that are the result of the abstract planning process, and input into any concrete planning. Providing a copy of the schema enables the parser to use the local copy instead of reaching out to the internet, and obtaining the latest version from the GriPhyN website dynamically.

4.3 vds.schema.ivr

Systems	all
Type	XML schema file location string
Default	${vds.home.sysconfdir}/iv-1.2.xsd

This file is a copy of the XML schema that describes invocation record files that are the result of the a grid launch in a remote or local site. Providing a copy of the schema enables the parser to use the local copy instead of reaching out to the internet, and obtaining the latest version from the GriPhyN website dynamically.

4.4 vds.db.pool

Moved to	vds.pool.file, section 4.6 (page 8)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

4.5 vds.db.tc

Moved to	vds.tc.file, section 4.7 (page 8)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

4.6 vds.pool.file

Systems	Pegasus
Type	file location string
Default	${vds.home.sysconfdir}/pool.config
Old name	vds.db.pool
See also	vds.pool.mode, section 8.1 (page 17)

Running things on the grid requires an extensive description of the capabilities of each compute cluster, commonly termed "pool". This property describes the location of the file that contains such a pool description. As the format is currently in flow, please refer to the userguide and Pegasus for details which format is expected.

4.7 vds.tc.file

Systems	all
Type	file location string
Default	${vds.home.localstatedir}/tc.data
Old name	vds.db.tc
See also	vds.tc.mode, section 8.2 (page 18)

The transformation catalog is a 3+ column textual file that describes in a simple column based format the mapping from a logical transformation for each pool to the physical application, and optional environment settings. All concrete planners use this repository to map the lTR from the abstract DAX into an application invocation. Refer to the user guide for details.

4.8 vds.db.rc

System	Chimera shell planner
Type	file-based RC file location string
Default	${vds.home.localstatedir}/rc.data

The shell planner that comes with Chimera completely by-passes any replica management catalogs that come with Pegasus. The shell planner uses a simple three column textual file to store the mapping of LFNs to PFNs and pools.

5 Chimera Vdc Access

5.1 vds.db.vdc.schema

System	Chimera VDC
Type	Java class name
Value[0]	SingleFileSchema
Value[1]	ChunkSchema
Value[2]	AnnotationSchema
Value[3]	NXDSchema
Default	org.griphyn.vdl.dbschema.SingleFileSchema
See also	vds.db.*.driver, section 5.5 (page 10)

This property denotes the schema that is being used to access a VDC. The VDC is a multi-layer architecture, with the dbschema sitting on top of the dbdrivers. Some schemas, like the default schema, do not access the driver level at all.

Currently available are only SingleFileSchema and ChunkSchema. These are names of Java classes. If no absolute Java classname is given, "org.griphyn.vdl.dbschema." is prepended. Thus, by deriving from the DatabaseSchema API, and implementing the VDC interface, users can provide their own classes here. Anything further properties in "vds.db.vdc.schema.*" will be copied into the schema properties, and thus made available to the schema.

SingleFileSchema This schema uses a file-based VDC. It is the default, as files are thought to be always available.

ChunkSchema This is a small VDC schema that uses an underlying rDBMS. Data outside the keys is stored in XML chunks, thus the name.

AnnotationSchema This is an extension of the ChunkSchema by providing annotation to filenames, transformations, derivations, formal arguments.

NXDSchema This is a schema storing XML into an native XML database. The database driver properties are not used.

5.2 vds.db.vdc.schema.xml.url

System	Chimera VDC, SingleFileSchema, ChunkSchema
Type	VDLx XML Schema location
Default	${vds.schema.vdl}

Various schemas manipulate VDLx internally, and need the knowledge of the schemas location to construct their parsers. This schema property enables users to provide backward compatible parsers for VDC data independent of the new data being read from files.

5.3 vds.db.vdc.schema.file.store

System	Chimera VDC, SingleFileSchema
Type	file location string
Default	${vds.home.localstatedir}/vdc.db

The VDC can be stored in a single file, which is the default mode of operation. This property determines the basename of the file. Extensions allow for a degree of rollbacks.

5.4 vds.db.ptc.schema

System	Chimera Provenance Tracking Catalog (PTC)
Type	Java class name
Value[0]	InvocationSchema
Value[1]	NXDInvSchema
Default	(no default)
See also	vds.db.*.driver, section 5.5 (page 10)

This property denotes the schema that is being used to access a PTC. The PTC is usually not a standard installation. If you use a database backend, you most likely have a schema that supports PTCs. By default, no PTC will be used.

Currently available is only InvocationSchema, if you want to store provenance tracking records. Beware, this can become a lot of data. The values are names of Java classes. If no absolute Java classname is given, "org.griphyn.vdl.dbschema." is prepended. Thus, by deriving from the DatabaseSchema API, and implementing the PTC interface, users can provide their own classes here. Anything further properties in "vds.db.ptc.schema.*" will be copied into the schema properties, and thus made available to the schema.

Alternatively, if you use a native XML database like eXist, you can store data using the NXDInvSchema. This will avoid using any of the other database driver properties.

5.5 vds.db.*.driver

System	Chimera VDC
Type	Java class name
Value[0]	Postgres
Value[1]	MySQL
Value[2]	SQLServer2000 (not yet implemented!)
Value[3]	Oracle (not yet implemented!)
Default	(no default)
See also	vds.db.vdc.schema, section 5.1 (page 9)
See also	vds.db.ptc.schema, section 5.4 (page 10)

The database driver class is dynamically loaded, as required by the schema. Currently, only PostGreSQL 7.3 and MySQL 4.0 are supported. Their respective JDBC3 driver is provided as part and parcel of the GVDS.

A user may provide their own implementation, derived from org.griphyn.vdl.dbdriver.DatabaseDriver, to talk to a database of their choice.

For each schema in vdc, ptc and tc, a driver is instantiated separately, which has the same prefix as the schema. This may result in multiple connections to the database backend. As fallback, the schema "*" driver is attempted.

5.6 vds.db.*.driver.url

System	Chimera VDC
Type	JDBC database URI string
Default	(no default)
Example	jdbc:postgresql:${user.name}

Each database has its own string to contact the database on a given host, port, and database. Although most driver URLs allow to pass arbitrary arguments, please use the vds.db.schema.driver.* keys or vds.db.*.driver.* to preload these arguments. THE URL IS A MANDATORY PROPERTY FOR ANY DBMS BACKEND.

```
Postgres : jdbc:postgresql:[//hostname[:port]/]database
MySQL    : jdbc:mysql://[hostname[:port]]/database
SQLServer: jdbc:microsoft:sqlserver://hostname:port
Oracle   : jdbc:oracle:thin:[user/password]@//host[:port]/service
```

5.7 vds.db.*.driver.user

System	Chimera VDC
Type	string
Default	(no default)
Example	${user.name}

In order to access a database, you must provide the name of your account on the DBMS. This property is database-independent. THIS IS A MANDATORY PROPERTY FOR MANY DBMS BACKENDS.

5.8 vds.db.*.driver.password

System	Chimera VDC
Type	string
Default	(no default)
Example	${user.name}

In order to access a database, you must provide an optional password of your account on the DBMS. This property is database-independent. THIS IS A MANDATORY PROPERTY, IF YOUR DBMS BACKEND ACCOUNT REQUIRES A PASSWORD.

5.9 vds.db.*.driver.*

System	Chimera VDC

Each database has a multitude of options to control in fine detail the further behaviour. You may want to check the JDBC3 documentation of the JDBC driver for your database for details. The keys will be passed as part of the connect properties by stripping the "vds.db.driver." prefix from them.

Postgres 7.3 parses the following properties:

```
vds.db.*.driver.user
vds.db.*.driver.password
vds.db.*.driver.PGHOST
vds.db.*.driver.PGPORT
vds.db.*.driver.charSet
vds.db.*.driver.compatible
```

MySQL 4.0 parses the following properties:

```
vds.db.*.driver.user
vds.db.*.driver.password
vds.db.*.driver.databaseName
vds.db.*.driver.serverName
vds.db.*.driver.portNumber
vds.db.*.driver.socketFactory
vds.db.*.driver.strictUpdates
vds.db.*.driver.ignoreNonTxTables
vds.db.*.driver.secondsBeforeRetryMaster
vds.db.*.driver.queriesBeforeRetryMaster
vds.db.*.driver.allowLoadLocalInfile
vds.db.*.driver.continueBatchOnError
vds.db.*.driver.pedantic
vds.db.*.driver.useStreamLengthsInPrepStmts
vds.db.*.driver.useTimezone
vds.db.*.driver.relaxAutoCommit
vds.db.*.driver.paranoid
vds.db.*.driver.autoReconnect
vds.db.*.driver.capitalizeTypeNames
vds.db.*.driver.ultraDevHack
vds.db.*.driver.strictFloatingPoint
vds.db.*.driver.useSSL
vds.db.*.driver.useCompression
vds.db.*.driver.socketTimeout
vds.db.*.driver.maxReconnects
vds.db.*.driver.initialTimeout
vds.db.*.driver.maxRows
vds.db.*.driver.useHostsInPrivileges
```

```
    vds.db.*.driver.interactiveClient
    vds.db.*.driver.useUnicode
    vds.db.*.driver.characterEncoding
```

MS SQL Server 2000 support the following properties (keys are case-insensitive, e.g. both "user" and "User" are valid):

```
    vds.db.*.driver.User
    vds.db.*.driver.Password
    vds.db.*.driver.DatabaseName
    vds.db.*.driver.ServerName
    vds.db.*.driver.HostProcess
    vds.db.*.driver.NetAddress
    vds.db.*.driver.PortNumber
    vds.db.*.driver.ProgramName
    vds.db.*.driver.SendStringParametersAsUnicode
    vds.db.*.driver.SelectMethod
```

6 Pegasus Replica Management Properties

6.1 vds.replica.mode

System	Pegasus
Type	enumeration
Value[0]	rls
Default	rls

The replica mode controls which replica management mechanism is being employed by Pegasus. The possible value at present is "rls".

6.2 vds.rls.url

System	Pegasus, RLS
Type	URI string
Default	(no default)

When using the modern RLS replica catalog, the URI to the RLI must be provided to Pegasus to enable it to look up filenames. There is no default.

6.3 vds.rls.exit

System	Pegasus, RLS
Type	enumeration
Value[0]	error
Value[1]	never
Default	error

The Cplanner code first queries the RLI to find the LRC urls and then queries each individual LRC's. If the mode is set to error (default) then the Cplanner will exit if it encounters any LRC to which it cannot authenticate or connect. If the mode is set to never the any LRC that cannot be authenticated or connected to is skipped unless if all of them fail then the CPlanner exits.

6.4 vds.rls.query

System	Pegasus, RLS
Type	enumeration
Value[0]	bulk
Value[1]	multiple
Default	bulk

The RLS server offers significant performance improvement when being bulk queried for filenames. Alas, bulk operations are only available starting with version 2.0.5 of the RLS software. The possible values for this key are "bulk" and "multiple". With the current RLS version being 2.0.7, "bulk" is the default. For backward compatibility, you can use "multiple".

6.5 vds.rls.query.attrib

System	Pegasus, RLS
Type	boolean
Value[0]	true
Value[1]	false
Default	false
See also	vds.transfer.mode.links, section 8.4 (page 20)
See also	vds.rls.query, section 6.4 (page 14)

Pegasus would prefer that each pfn be associated with a pool attribute that lets it associate a pfn with a gvds pool. This can be used in conjunction with vds.transfer.mode.links and multiple mode of vds.transfer.mode to create symbolic links to the pfn's. This is only when a pfn is found to be at the same execution pool where Pegasus has sceduled the job. A point to be taken care of is that, this should be used if the query mode to rls is bulk in order to enable bulk query of attributes in RLS. This bulk query of attributes in RLS is available from version 2.0.9 or later.

7 Pegasus Site Selection Properties

7.1 vds.site.selector.mode

Moved to	vds.site.selector, section 7.2 (page 15)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

7.2 vds.site.selector

System	Pegasus
Since	1.2.8
Type	enumeration
Value[0]	Random
Value[1]	RoundRobin
Value[2]	NonJavaCallout
Default	Random

The site selection in Pegasus can be on basis of any of the following strategies.

1. Pegasus assigns jobs randomly amongst the sites that can execute them.

2. Pegasus assigns jobs in a round robin manner amongst the sites that can execute them. Since each site cannot execute everytype of job, the round robin scheduling is done per level on a sorted list. The sorting is on the basis of the number of jobs a particular site has been assigned in that level so far. If a job cannot be run on the first site in the queue (due to no matching entry in the transformation catalog for the transformation referred to by the job), it goes to the next one and so on. This implementation defaults to classic round robin in the case where all the jobs in the workflow can run on all the sites.

3. Pegasus can also callout to an external site selector. In this mode a temporary file is prepared containing the job information that is passed to the site selector as an argument while invoking it. The path to the site selector is specified by setting the property vds.site.selector.path. The environment variables that need to be set to run the site selector can be specified using the properties with a vds.site.selector.env. prefix. The temporary file contains information about the job that needs to be scheduled. It contains key value pairs with each key value pair being on a new line and separated by a =.

Currently the following keys are supported by the site selector:

vds_job_name is the unique id of the job in the current workflow.
vds_transformation is the VDLt-style specification of a given transformation, using the triple of namespace :: logicalname : version.
vds_scheduler_preference denotes a jobmanager, either fork or regular.
vds_exec_pools is a comma separated list of execution pools.
vds_input_files is a comma separated list of logical names of input files for the jobs. If there are no input files then the value is set to NONE.

7.3 vds.site.selector.path

System	Pegasus
Since	1.2.3
Type	String

If one calls out to an external site selector using the NonJavaCallout mode, this refers to the path where the site selector is installed. In case other strategies are used it does not need to be set.

7.4 vds.site.selector.env.*

System	Pegasus
Since	1.2.3
Type	String

The environment variables that need to be set while callout to the site selector. These are the variables that the user would set if running the site selector on the command line. The name of the environment variable is got by stripping the keys of the prefix "vds.site.selector.env." prefix from them. The value of the environment variable is the value of the property.

e.g vds.site.selector.path.LD_LIBRARY_PATH /globus/lib would lead to the site selector being called with the LD_LIBRARY_PATH set to /globus/lib.

7.5 vds.site.selector.timeout

System	Pegasus
Since	1.3.0
Type	non negative integer
Default	60

It sets the number of seconds Pegasus waits to hear back from an external site selector using the NonJavaCallout interface before timing out.

7.6 vds.site.selector.keep.tmp

System	Pegasus
Since	1.3.2
Type	enumeration
Value[0]	onerror
Value[1]	always
Value[2]	never
Default	onerror

It determines whether Pegasus deletes the temporary input files that are generated in the temp directory or not. These temporary input files are passed as input to the external site selectors.

A temporary input file is created for each that needs to be scheduled.

8 Pegasus In Action

8.1 vds.pool.mode

System	Pegasus, MDS, PoolConfig
Type	enumeration
Value[0]	single
Value[1]	multiple
Value[2]	xml
Default	single

The pool.config file is available in two major flavors:

1. The old textual format employs multiple columns in a textual file, see sample.pool.config.old. This format is DEPRECATED and will be removed in the mid-term future. The odl textual format is available in two parsing modes for scalability purposes:

   a) The "single" modes reads the old format pool.config file once and only once. The data is kept in memory for better performance. We recommend this mode for pool files with less than 1k lines.

   b) The "multiple" mode reads of the old format pool.config file every time it needs to look up somethings. Use this mode if you have memory problems, or if you have an unusually huge pool.config file.

2. The "xml" format is an XML-based file. It is generated using the genpoolconfig client application program that is shipped with Pegasus. The XML input file for Pegasus can be generated in various ways, that can be used exclusively or combined at your option:

a) The pool configuration file can be generated using information that is published in MDS (see VDS user guide document). This is the recommended way.

b) It can also be published by converting the new, easier to read and modify local multiline pool config file. An example is provided in sample.pool.config.new. Use this option if you have no network connectivity, or for tests.

8.2 vds.tc.mode

System	Pegasus, TC
Type	enumeration
Value[0]	single
Value[1]	multiple
Value[2]	OldFile
Value[3]	File
Value[4]	Database
Default	File
See also	vds.db.tc, section 4.5 (page 7)

The Transformation Catalog (TC) is currently a three-column file. Similar to the pool configuration, for speed and memory consumption, two parse modes are available:

1. In "single", "multiple" and "OldFile" mode, the old TC file is read once and cached in main memory. This yields a good performance while consuming some memory. We recommend to use this methods for files with less than 10000 entries.

   The old file format consists of for tabulator-separated columns: The resource identifier, the logical transformation name, the physical installation path to the application, and optional environment variables or the string `null`. The old format understands the logical transformation specification in both, the ancient format using underscores ns__id_vs and the modern colonized version ns::id:vs.

   The old format has been DEPRECATED. Please use the conversion tool in `contrib/tc-converter/tc-old2` to convert to the new format or run

   ```
   tc-client -Dvds.tc.mode=single -q B > tc.data.net
   ```

2. In "File" mode, the new file format is understood. The file is read and cached in memory. Any modifications, as adding or deleting, causes an update of the memory and hence to the file underneath. All queries are done against the memory representation. The new TC file format uses 6 columns:

   a) The resource ID is represented in the first column.

   b) The logical transformation uses the colonized format ns::name:vs.

   c) The path to the application on the system

d) The installation type is identified by one of the following keywords - all upper case: INSTALLED, STATIC_BINARY, DYNAMIC_BINARY, SCRIPT. If not specified, or `NULL` is used, the type defaults to INSTALLED.

e) The system is of the format ARCH::OS[:VER:GLIBC]. The Following arch types are understood: "INTEL32", "INTEL64", "SPARCV7", "SPARCV9". The Following os types are understood: "LINUX", "SUNOS", "AIX". If unset or `NULL`, defaults to INTEL32::LINUX.

f) Profiles are written in the format NS::KEY=VALUE,KEY2=VALUE;NS2::KEY3=VALUE3 Multiple key-values for same namespace are seperated by a comma "," and multiple namespaces are seperated by a semicolon ";". If any of your profile values contains a comma you must not use the namespace abbreviator.

3. In "Database" mode, the transformation catalog is kept in a relational database. Currently only the mysql DB is supported correctly. To set up the the database, use the schema in `$VDS_HOME/sql/create-my-tc`

Future modifications to the TC may extend the enumeration. To implement your own TC implementation see org.girphyn.cPlanner.tc.TCMechanism. To load the class set vds.tc.mode to the TC implementation class.

8.3 vds.transfer.mode

System	Pegasus
Type	enumeration
Value[0]	single
Value[1]	multiple
Value[2]	T2
Value[3]	StorkSingle
Default	single

Each job usually has data products that are required to be staged out to a final resting place, or staged to another job running at a different pool. The transfer mode determines the number of transfer jobs added to a compute job:

1. In "single" mode each file that needs to be transferred will generate one transfer job. This mode works well with kickstart, and employs globus-url-copy as transfer agent. It uses two party transfers based on the assumption that the gridftp server runs on the gatekeeper.

2. In "multiple" mode, all files will be attempted to be transferred with just one transfer job, resulting in multiple transfers. This mode uses the provided appliation "transfer" that internally invokes globus-url-copy to transfer each file. "transfer" may not agree with "kickstart", though this is to be fixed. Files are transferred sequentially, one after the other.

3. In "T2" mode, like the "multiple" mode all files will be attempted to be transfered with just one transfer job. In addition, T2 can handle conditional transfers, whereby it signals a success in case

of failures during transfers. The input file for the transfer job that is constructed contains multiple source and destination urls for the same transfer. The transfer fails only if all pair candidates are attempted without success.

4. In "StorkSingle" mode, the transfers are scheduled using Stork and the transfer submit files are generated in stork format. This is an experimental mode that is being tested currently.

Pegasus is researching a new transfer job that will enable multiple transfers in parallel. Further properties will allow the parallelization to be tuned.

8.4 vds.transfer.mode.links

System	Pegasus
Type	boolean
Default	false
See also	vds.transfer.mode, section 8.3 (page 19)
See also	vds.transfer.mode.force, section 8.5 (page 20)
See also	vds.rls.query.attrib, section 6.5 (page 14)

If this is set, and the transfer mode is set to multiple i.e. using the transfer executable distributed with the VDS. On setting this property, if Pegasus while fetching data from the RLS sees a pool attribute associated with the pfn that matches the execution pool on which the data has to be transferred to, Pegasus instead of the url returned by the RLS replaces it with a file based url. This supposes that the if the pools match the filesystems are visible to the remote execution directory where input data resides. On seeing both the source and destination urls as file based url's the transfer executable spawns a job that creates a symbolic link by calling ln -s on the remote pool. This ends up bypassing the grid ftp server and reduces the load on it, and is much faster.

8.5 vds.transfer.mode.force

System	Pegasus
Type	unknown
Default	unknown

This needs to be documented!

8.6 vds.transfer.throttle.processes

System	Pegasus
Type	integer
Default	4
See also	vds.transfer.mode, section 8.3 (page 19)
See also	vds.transfer.throttle.streams, section 8.7 (page 21)

This property is picked up when transfer mode (vds.transfer.mode) is multiple. In this mode, multiple files are transferred using a transfer executable that comes with the VDS system. This transfer executable attempts to transfer multiple files by spawning multiple g-u-c processes. By default a maximum of 4 processes are spawned to transfer the files. Using this one can change the number of processes that are spawned by the transfer executable.

8.7 vds.transfer.throttle.streams

System	Pegasus
Type	integer
Default	1
See also	vds.transfer.mode, section 8.3 (page 19)
See also	vds.transfer.throttle.processes, section 8.6 (page 20)

Whatever the transfer mode specified, at present each uses globus-url-copy (g-u-c) as the underlying transfer mechanism. g-u-c can open multiple streams to do the ftp data transfer. This property can be used to set the number of streams that are used to transfer one file by underlying g-u-c. It directly maps to the -p option of g-u-c.

8.8 vds.transfer.force

System	Pegasus
Type	boolean
Default	false
See also	vds.transfer.mode, section 8.3 (page 19)
See also	vds.transfer.mode.links, section 8.4 (page 20)

If this is set, the underlying transfer mechanism should use the force option if available while transferring the files. At present if the vds.transfer.mode is set to multiple and vds.transfer.mode.links to true the force option would be set, that is used while creating the symbolic links on the execution pools.

8.9 vds.transfer.thirdparty.pools

System	Pegasus
Type	comma separated list of pools
Default	no default
since	1.2.0

By default Pegasus employs the push/pull model of transfer for transferring files in and out of a pool. It does use the third party transfer option that grid ftp servers provide. This list specifies the list of pools on which the user wants third party transfers instead of the normal mode. Normally for a push transfer the source url is file://blah and destination url is gsiftp://blah. However in case of a third party pool both the urls would be gsiftp://blah. For all these pools, the transfers are actually scheduled on the submit host (pool local) in the scheduler universe.

8.10 vds.exitcode.mode

System	Pegasus
Type	enumeration
Value[0]	all
Value[1]	none
Value[2]	essential
Default	none
See also	vds.exitcode.arguments, section 8.11 (page 22)

Jobs that are "kickstarted" by a grid launcher report back information about the execution, resource consumption, and - most importantly - the exit code of the remote application. Armed with this knowledge, it is possible to have DAGMan stop the workflow and create a rescue workflow on remote execution errors. In "all" mode, each kickstarted job's invocation record will be parsed by a DAGMan postscript. In "none" mode, the default, remote failures will not abort the DAG. In "essential" mode, only certain classes of remote jobs get the ability to abort the workflow, while other, non-essential, classes of jobs (at present the replica registration jobs) will not have their invocation record abort the workflow.

8.11 vds.exitcode.arguments

System	Pegasus
Type	string
Default	no default
since	1.2.14
See also	vds.exitcode.mode, section 8.10 (page 22)

This specifies the arguments by which the exitcode is invoked on the kickstart output of a job. It applies to all the jobs, for which exitcode is invoked as determined by vds.exitcode.mode property.

8.12 vds.dir.exec

System	Pegasus
Type	remote directory location string
Default	(no default)

This property modifies the remote location work directory in which all your jobs will run. If the path is relative then it is appended to the exec mount point (old pool configuration) or work directory (new pool configuration), as specified in the pool config file. If the path is absolute then it overrides the mount point specified in the pool config file.

8.13 vds.dir.storage

System	Pegasus
Type	remote directory location string
Default	(no default)

This property modifies the remote storage location on various pools. If the path is relative then it is appended to the storage mount point specified in the pool.config file. If the path is absolute then it overrides the storage mount point specified in the pool config file.

8.14 vds.dir.create.mode

System	Pegasus
Type	enumeration
Value[0]	HourGlass
Value[1]	Tentacles
Default	HourGlass

If the `--randomdir` option is given to the Planner at runtime, the Pegasus planner ends up adding nodes that end up creating the random directories at the remote pool sites, before any jobs are actually run. The two modes end up determining the placement of these nodes and their dependencies to the rest of the graph.

HourGlass It adds a make directory node at the top level of the graph, and all these concat to a single dummy job before branching out to the root nodes of the original/ concrete dag so far. So we end up introducing a classic X shape at the top of the graph. Hence the name HourGlass.

Tentacles This ends up placing the create directory jobs at the top of the graph. However instead of constricting it to an hour glass shape, this mode links it to all the relevant nodes for which the create dir job is necessary. It is like that it spreads it's tentacleas all around. This ends up putting more load on the DagMan with all the dependencies but removes the restriction of the plan progressing only when all the create directory jobs have progressed on the remote pools, as in the HourGlass model.

8.15 vds.lsf.projects

Moved to	vds.scheduler.remote.queues, section 8.17 (page 24)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

8.16 vds.scheduler.remote.projects

System	Pegasus
Type	list of kv pairs
Default	(no default)
Example	jazz=PDQ,pnnl=emsl12491

This property allows to set the *project* name that is to be used for each pool. Usually, such project names are specific to a small set of users, and can not be well set in the pool configuration file. Please note that the key is the pool handle, and the value is the project name. Setting the project will result in the generation of an RSL term (project=xxxx) for the matching pool handle.

8.17 vds.scheduler.remote.queues

System	Pegasus
Type	list of kv pairs
Default	(no default)
Example	jazz=PDQ,pnnl=emsl12491

This property allows to set the *queue* name that is to be used for each pool. Usually, such queue names are specific to single users, and can thus not be well set in the pool configuration file. Please note that the key is the pool handle, and the value is the queue name. The property is applicable to any remote scheduling system that employs named queues, e.g. PBS or LSF. Setting the queue will result in the generation of an RSL clause (queue=xxxx) for the matching pool handle.

8.18 vds.scheduler.remote.maxwalltimes

System	Pegasus
Type	list of kv pairs
Default	(no default)
Example	jazz=10,pnnl=20

This property allows to set the *walltime* for your jobs on each pool. Max Walltime means the maximum amount of time a job would run for on a pool. This property is applicable to any remote scheduling system that employs walltimes like PBS. Setting the walltime will result in the generation of an RSL clause (maxwalltime=xxxx) for the matching pool handle. Please note that most if all scheduling systems that use this kill the job if the jobs running time exceed the advertised walltime in the job description.

8.19 vds.pegasus.kickstart-condor

Moved to	vds.scheduler.condor.start, section 8.22 (page 25)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

8.20 vds.pegasus.condor-bin

Moved to	vds.scheduler.condor.bin, section 8.23 (page 25)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

8.21 vds.pegasus.condor-config

Moved to	vds.scheduler.condor.config, section 8.24 (page 26)

This property name was deprecated, and will eventually be phased out. Please use the new name in the future.

8.22 vds.scheduler.condor.start

System	Pegasus, DAG auto-submit
Type	local file location string
Default	(no default)
Old name	vds.pegasus.kickstart-condor
See also	vds.scheduler.condor.bin, section 8.23 (page 25)
See also	vds.scheduler.condor.config, section 8.24 (page 26)

This property needs to be set if you intend to submit jobs to Condor from Pegasus itself instead of stopping after all Condor files are generated. Starting a generated workflow requires to activate condor_submit_dag in the DAG directory. An implementation for the auto-submittor for this option is provided in ${vds.home}/bin/kickstart-condor

8.23 vds.scheduler.condor.bin

System	Pegasus, DAG auto-submit
Type	local directory location string
Default	(no default)
Old name	vds.pegasus.condor-bin
See also	vds.scheduler.condor.start, section 8.22 (page 25)
See also	vds.scheduler.condor.config, section 8.24 (page 26)

This property needs to be set if you intend to submit jobs to Condor from Pegasus itself instead of stopping after all Condor files are generated. This property points to the location of the "bin" directory of your local Condor installation.

8.24 vds.scheduler.condor.config

System	Pegasus
Type	local file location string
Default	(no default)
Old name	vds.pegasus.condor-config
See also	vds.scheduler.condor.start, section 8.22 (page 25)
See also	vds.scheduler.condor.bin, section 8.23 (page 25)

This property needs to be set if you intend to submit jobs to Condor from Pegasus itself instead of stopping after all Condor files are generated. This property points to the location where the condor_config file is stored. Refer to the Condor manual for the default locations of the Condor system.

8.25 vds.scheduler.condor.release

System	Pegasus
Type	non-negative integer
Default	3

This property determines the number of release attempts that are written into the submit file. Condor will hold jobs on certain kinds of failures. Many known failing conditions are transient, thus, if the job is automatically release after a short time, it usually progresses.

8.26 vds.scheduler.condor.remove

System	Pegasus
Type	non-negative integer
Default	3
See also	vds.scheduler.condor.release, section 8.25 (page 26)
Since	1.3.0

This property determines the number of hold attemts that happen before Condor removes the job from the queue. This value is tied to the number of release attempts. Hence, Pegasus enforces release > remove >= 0. A value of zero indicates, that the user does not want the job to be removed from the queue. In that case, no periodic_remove statement would be constructed. If the property is not set, the value of vds.scheduler.condor.release is used. If both the properties are unset, then the default value of 3 is used.

8.27 vds.scheduler.condor.retry

System	Pegasus
Type	non-negative integer
Default	3
Since	1.3.0

This property determines the number of attempts DAGMAN makes to execute a job in case of failure. In case of deferred planning this can be used to do replanning. A failure during the execution of a partition can trigger the execution of Pegasus via a prescript, that may result in a different plan for the partition being generated and executed. Note, this is different from vds.scheduler.condor.release which results in condor retrying the same job (submit file) when a job goes in HOLD state.

8.28 vds.scheduler.condor.output.stream

System	Pegasus
Type	Boolean
Value[0]	false
Value[1]	true
Default	true
See also	vds.scheduler.condor.error.stream, section 8.29 (page 27)

By default, GASS streams back stdout continuously. In this default mode, it will require additional filedescriptors on the gatekeeper. Recent versions of Globus and Condor allow the content of stdout to be streamed back after the job is done. While no content is visible during the execution of the job, it saves precious gatekeeper resources.

8.29 vds.scheduler.condor.error.stream

System	Pegasus
Type	Boolean
Value[0]	false
Value[1]	true
Default	true
See also	vds.scheduler.condor.output.stream, section 8.28 (page 27)

By default, GASS streams back stderr continuously. In this default mode, it will require additional filedescriptors on the gatekeeper. Recent versions of Globus and Condor allow the content of stderr to be streamed back after the job is done. While no content is visible during the execution of the job, it saves precious gatekeeper resources.

8.30 vds.giis.host

System	Pegasus
Type	string
Default	(no default)
See also	vds.pool.mode, section 8.1 (page 17)

This property needs to be set if you set the vds.pool.mode to xml and you wish to obtain the pool config information dynamically using MDS along with or without local pool config support files. The property needs to be set to the hostname and optional port of the GIIS service that is aggregating pool information, e.g. smarty.isi.edu:2135

8.31 vds.giis.dn

System	Pegasus
Type	X.400 string
Default	(no default)
See also	vds.pool.mode, section 8.1 (page 17)

If the vds.giis.host is set, and MDS is used to obtain the pool configuration, the ’distinguished name’ to contact the GIIS needs to be specified with this property, e.g. "MDS-vo-name=GRIPHYN,o=Grid"

8.32 vds.log4j.log

System	Pegasus, LoggerService, Log4j
Type	filename
Default	(no default)
See also	vds.loggerservice.url, section 8.33 (page 28)

If you plan to log the interal RLS queries in Pegasus to a local file using log4j then set the property to the location of the file you want the log written to. THIS IS AN EXPERIMENTAL FEATURE.

8.33 vds.loggerservice.url

System	Pegasus, Ogsa LoggerService
Type	URI string
Default	(no default)
See also	vds.log4j.log, section 8.32 (page 28)

If you plan to log internal RLS queries in pegasus to an OGSA logger service. Set the property to the url of the logger service. THIS IS AN EXPERIMENTAL FEATURE.

9 Chimera In Action

9.1 vds.log.*

System	ChimWarning: 85: Found regular textual paragraph, copying era
Type	filename or stdio handle
Default	(no default)
Example	vds.log.chunk=stderr

The Chimera system has (currently, about to change) a logging system that works with queues and levels. Each logging Q can be addressed separately, and given either a filename to append log information onto, "stdout" for standard output, or "stderr" for standard error. You may use the same filename for different queues.

9.2 vds.verbose

System	Chimera
Type	flag
Default	(not specified)

If the verbose option is specified, the Chimera system will provide maximum logging on all queues. This is very verbose, but sometimes the easiest way to track an error.

10 Interface To Dagman

The Condor DAGMan facility is usually activate using the condor_submit_dag command. However, many shapes of workflows have the ability to either overburden the submit host, or overflow remote gatekeeper hosts. While DAGMan provides throttles, unfortunately these can only be supplied on the command-line. Thus, the GVDS provides a versatile wrapper to invoke DAGMan, called vds-submit-dag. It can be configured from the command-line, from user- and system properties, and by defaults.

10.1 vds.dagman.maxpre

System	DAGman wrapper
Type	integer
Default	20
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html

The vds-submit-dag wrapper processes properties to set DAGMan commandline arguments. The argument sets the maximum number of PRE scripts within the DAG that may be running at one time. If this option is set to 0, the default number of PRE scripts is unlimited. The GVDS system throttles artificially to a maximum of 20 PRE scripts.

10.2 vds.dagman.maxpost

System	DAGman wrapper
Type	integer
Default	20
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html

The vds-submit-dag wrapper processes properties to set DAGMan commandline arguments. The argument sets the maximum number of POST scripts within the DAG that may be running at one time. If this option is set to 0, the default number of POST scripts is unlimited. The GVDS system throttles artificially to a maximum of 20 POST scripts.

10.3 vds.dagman.maxjobs

System	DAGman wrapper
Type	integer
Default	200
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html

The vds-submit-dag wrapper processes properties to set DAGMan commandline arguments. The argument sets the maximum number of jobs within the DAG that will be submitted to Condor at one time. If the option is omitted, the default number of jobs is unlimited. The GVDS system throttles artificially to a maximum of 200 simultaneous jobs that are visible to the Condor system. You may want to raise this bar.

vds.dagman.maxjobs 200

10.4 vds.dagman.nofity

System	DAGman wrapper
Type	Case-insensitive enumeration
Value[0]	Complete
Value[1]	Error
Value[2]	Never
Default	Error
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit.html

The vds-submit-dag wrapper processes properties to set DAGMan commandline arguments. The argument sets the e-mail notification for DAGMan itself. This information will be used within the Condor submit description file for DAGMan. This file is produced by the the condor_submit_dag. See notification within the section of submit description file commands in the condor_submit manual page for specification of value. Many users prefer the value NEVER.

10.5 vds.dagman.verbose

System	DAGman wrapper
Type	Boolean
Value[0]	false
Value[1]	true
Default	false
Document	http://www.cs.wisc.edu/condor/manual/v6.6/condor_submit_dag.html

The vds-submit-dag wrapper processes properties to set DAGMan commandline arguments. If set and true, the argument activates verbose output in case of DAGMan errors.

Index